Opt-AI.

Weekly Seminar

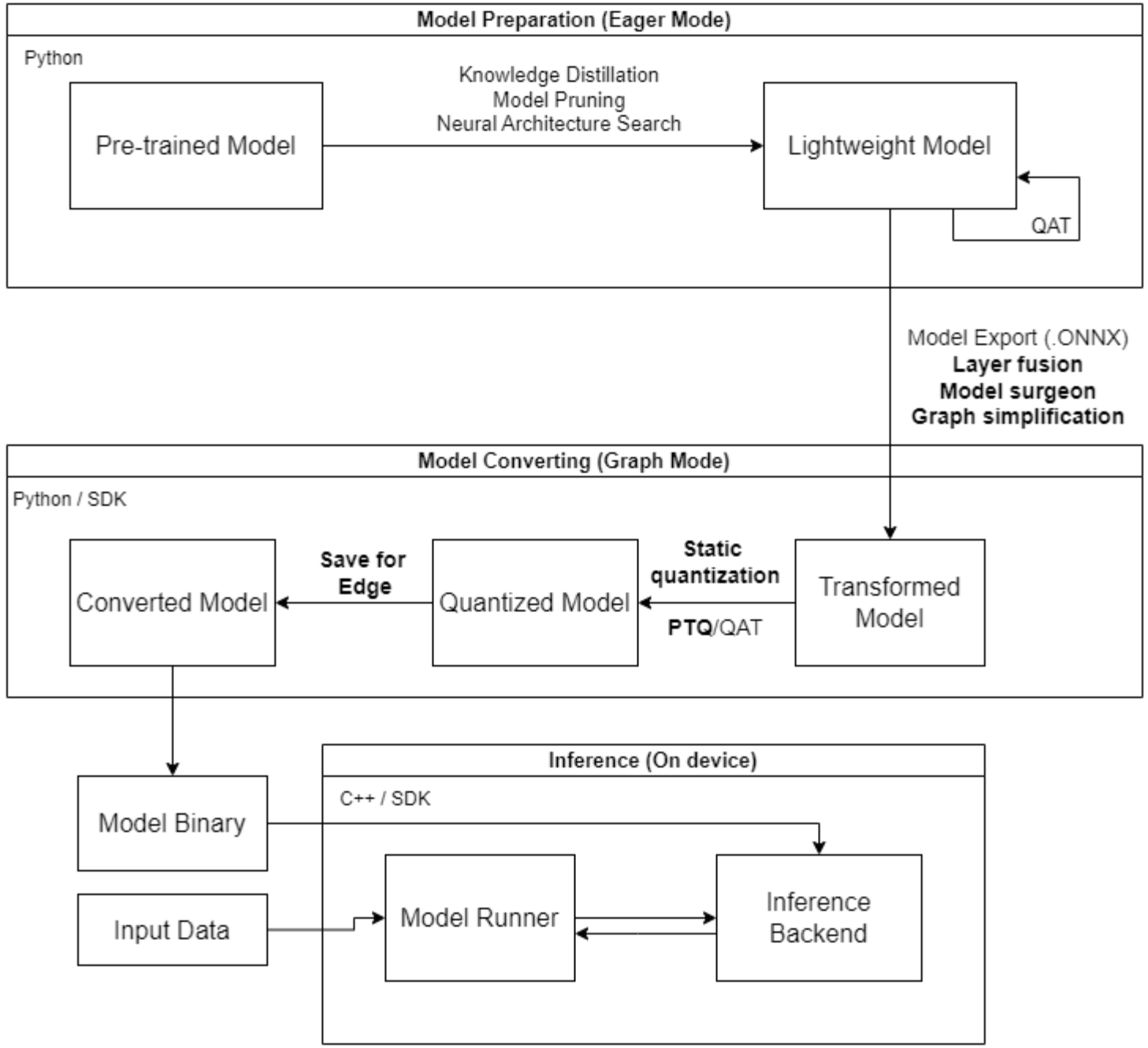
Opt-AI. LLM Research Team

Llama-3.2-1B QNN Experiments

Table of Contents

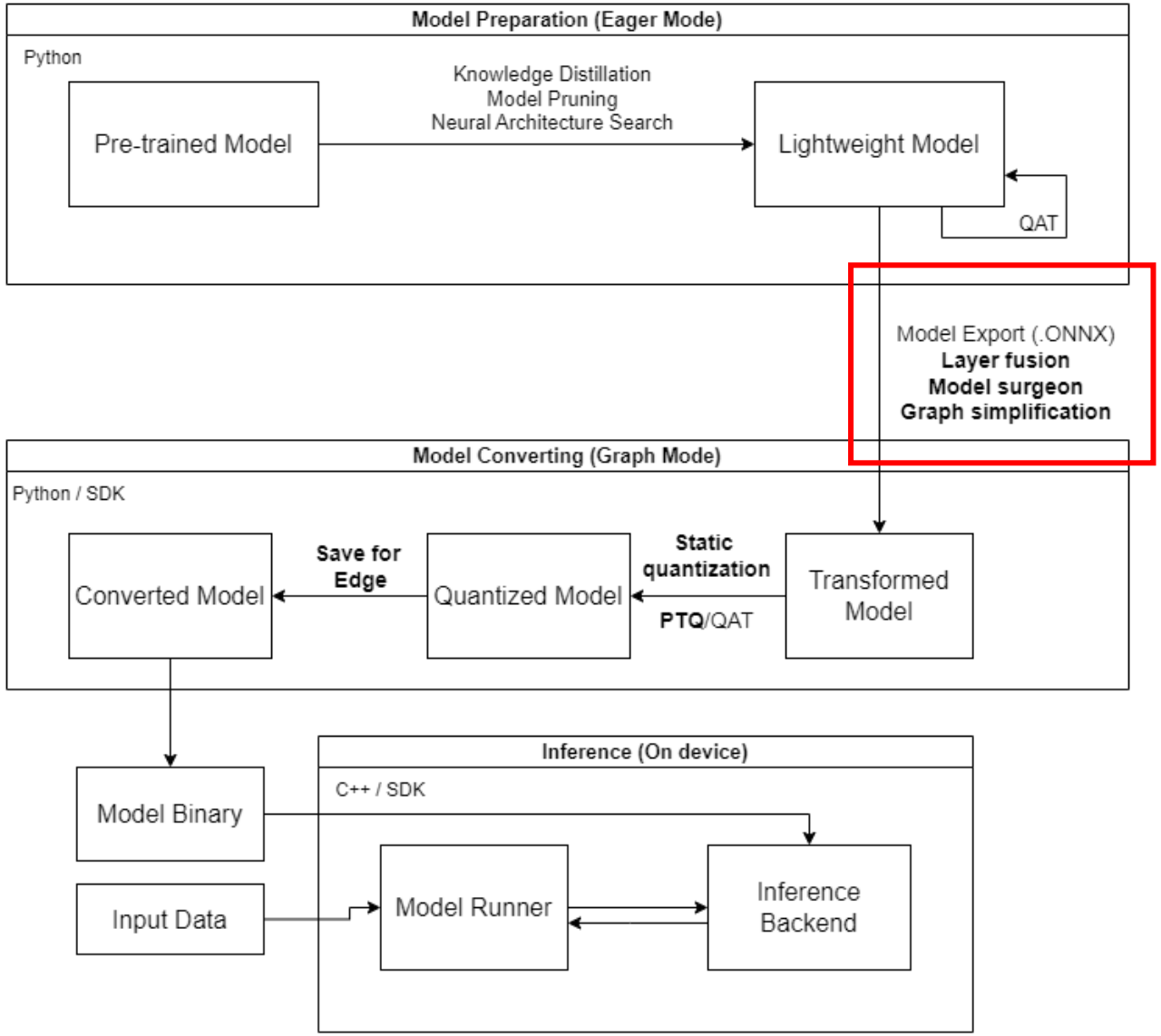
01
Review Last Lessons

Quiz

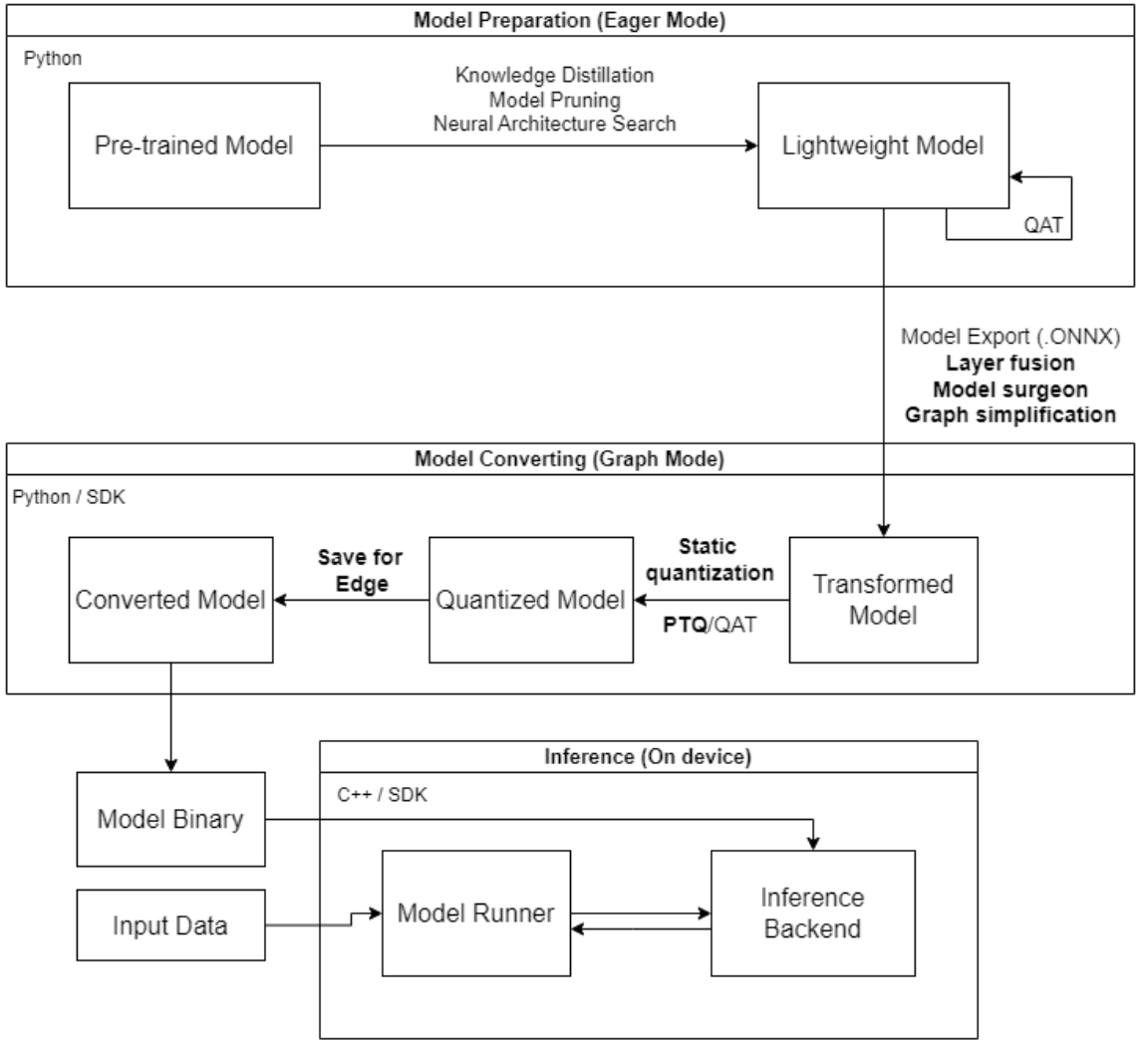

Model Preparation (Eager Mode)
Python
Pre-trained Model
Knowledge Distillation
Model Pruning
Neural Architecture Search
Lightweight Model
QAT
Model Export (.ONNX)
Layer fusion
Model surgeon
Graph simplification
Model Converting (Graph Mode)
Python / SDK
Converted Model
Save for Edge
Quantized Model
Static quantization
PTQ/QAT
Transformed Model
Inference (On device)
C++ / SDK
Model Binary
Input Data
Model Runner
Inference Backend

```python
def main() -> None:
    seed = 42
    torch.manual_seed(seed)
    modelname = "llama2"
    parser = build_args_parser()
    args = parser.parse_args()
    export_llama(modelname, args)
```

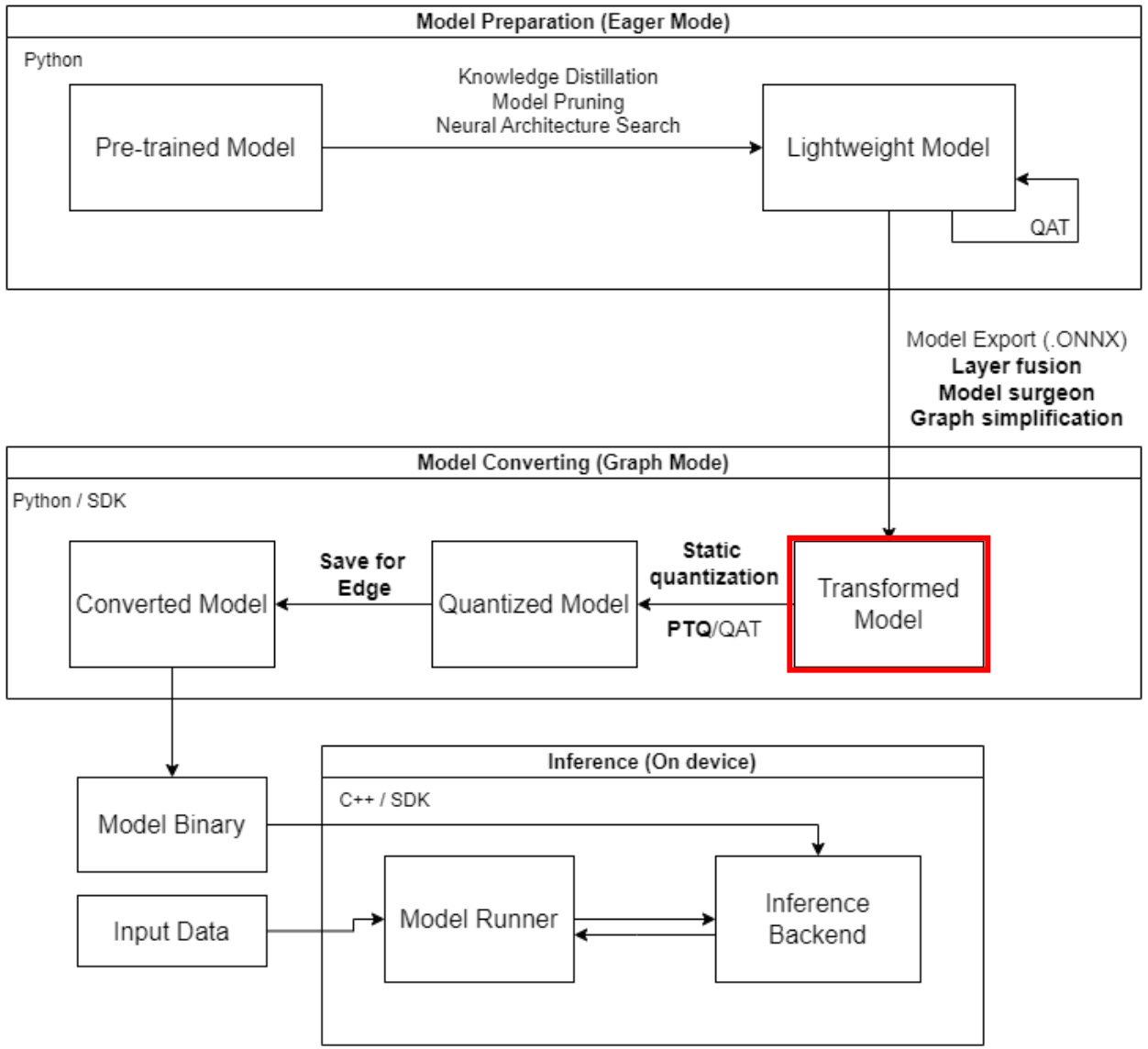
```python
def export_llama(modelname, args) -> str:
    """
    others code..
    """
    else:
        builder = _export_llama(modelname, args)
        assert (
            filename := builder.get_saved_pte_filename()
        ) is not None, "Fail to get file name from builder"
        return filename
```



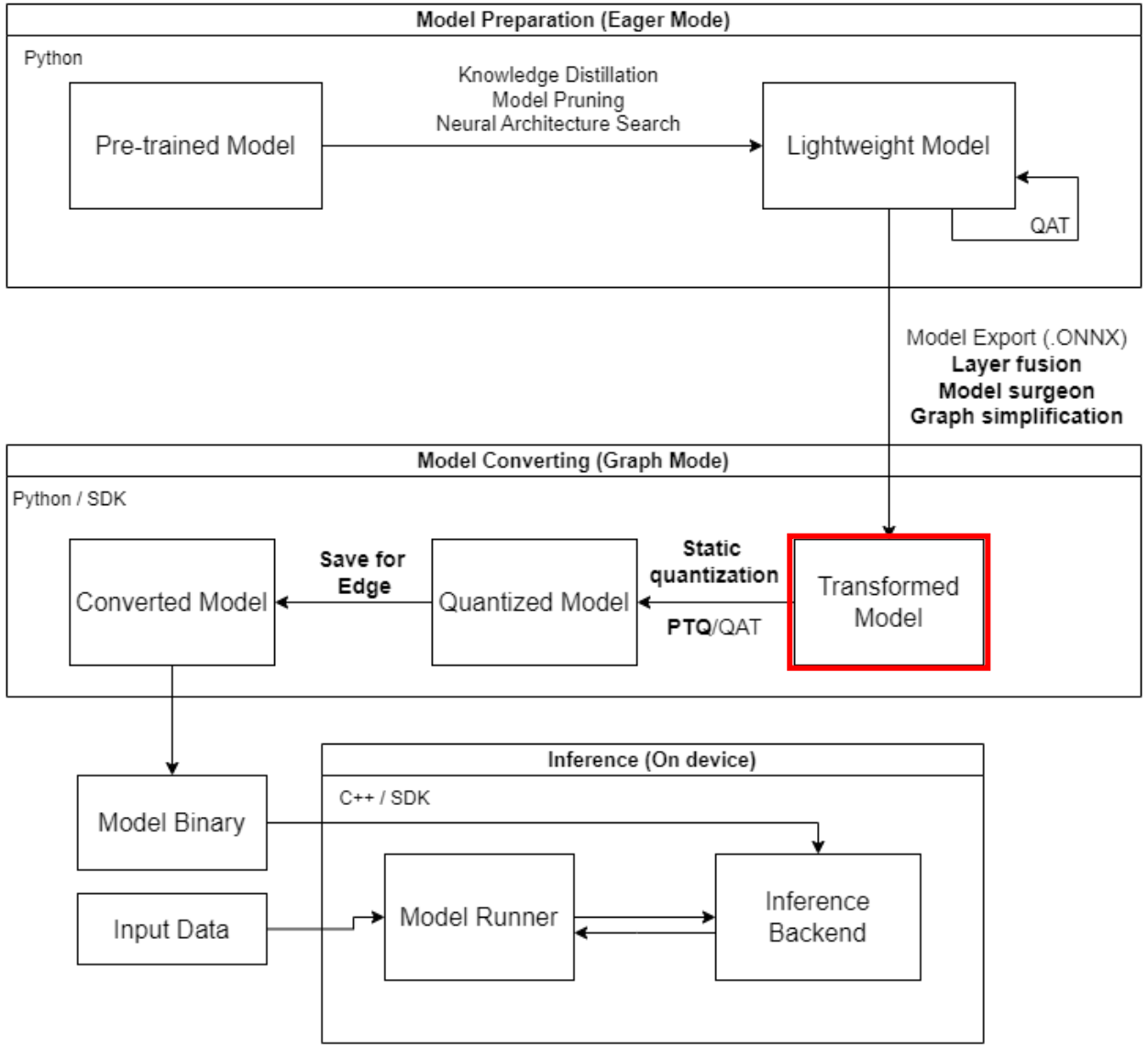
```python
def _export_llama(modelname, args) -> LLMEdgeManager:  # noqa: C901
    _validate_args(args)
    pt2e_quant_params, quantizers, quant_dtype = get_quantizer_and_quant_params(args)

    # export_to_edge
    builder_exported_to_edge = (
        _prepare_for_llama_export(modelname, args)
        .capture_pre_autograd_graph()
        .pt2e_quantize(quantizers)
        .export_to_edge()
    )
```



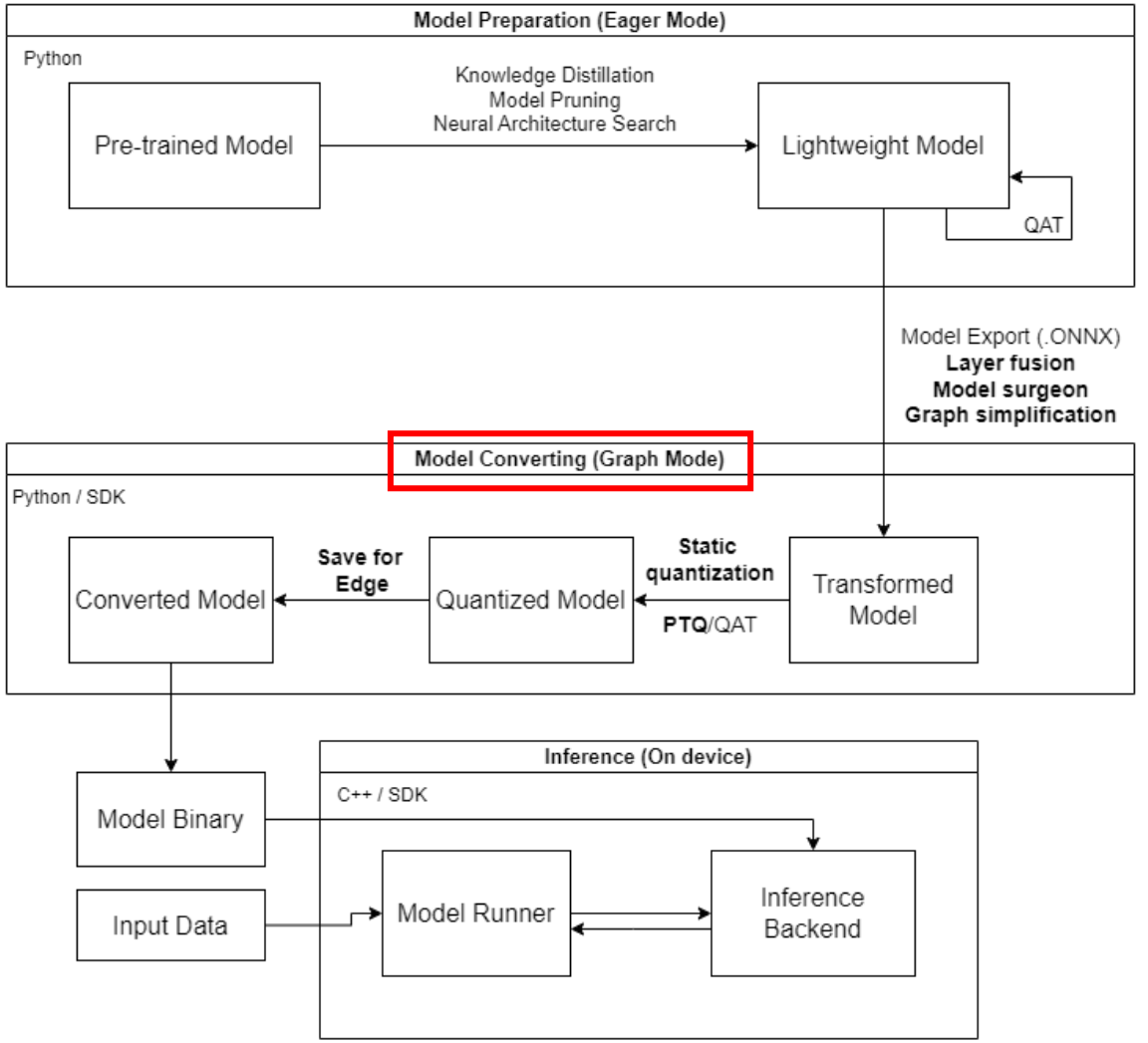
```python
def _prepare_for_llama_export(modelname: str, args) -> LLMEdgeManager:
    """
    Helper function for export_llama. Loads the model from checkpoint and params,
    and sets up a LLMEdgeManager with initial transforms and dtype conversion.

    Returns a LLMEdgeManager prior to calling export_to_edge with quantizers
    """

    # load model from checkpoint and params.json
    checkpoint_path = canonical_path(args.checkpoint) if args.checkpoint else None
    checkpoint_dir = (
        canonical_path(args.checkpoint_dir) if args.checkpoint_dir else None
    )
    params_path = canonical_path(args.params)
    output_dir_path = canonical_path(args.output_dir, dir=True)
    weight_type = WeightType.FAIRSEQ2 if args.fairseq2 else WeightType.LLAMA

    return (
        _load_llama_model(
            modelname=modelname,
            checkpoint=checkpoint_path,
            checkpoint_dir=checkpoint_dir,
            params_path=params_path,
            use_kv_cache=args.use_kv_cache,
            use_sdpa_with_kv_cache=args.use_sdpa_with_kv_cache,
            generate_full_logits=args.generate_full_logits,
            weight_type=weight_type,
            enable_dynamic_shape=args.enable_dynamic_shape,
            calibration_tasks=args.calibration_tasks,
            calibration_limit=args.calibration_limit,
            calibration_seq_length=args.calibration_seq_length,
            calibration_data=args.calibration_data,
            tokenizer_path=args.tokenizer_path,
            verbose=args.verbose,
            max_seq_len=args.max_seq_length,
            metadata_str=args.metadata,
            args=args,
        )
        .set_output_dir(output_dir_path)
        .to_dtype(dtype_override)
        .source_transform(_get_source_transforms(modelname, dtype_override, args))
    )
```



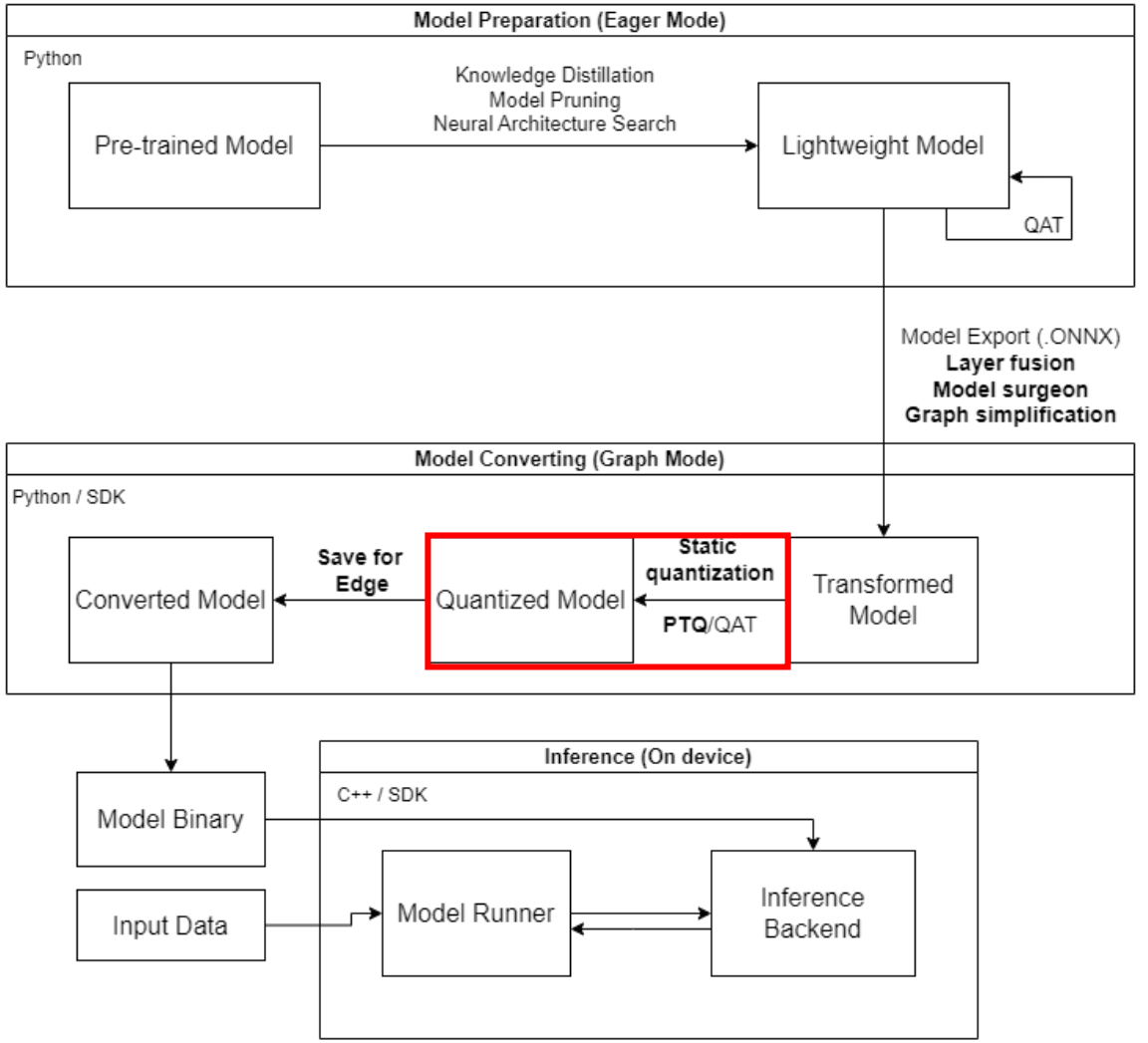
```python
if args.use_kv_cache:
        if args.qnn:
            from executorch.backends.qualcomm.utils.utils import (
                convert_linear_to_conv2d,
            )

            if args.use_qnn_sha:
                if args.optimized_rotation_path:
                    transforms.append(fuse_layer_norms)
                    transforms.append(
                        get_model_with_r1_r2(args.optimized_rotation_path)
                    )
                transforms.append(replace_attention_to_attention_sha)
                transforms.append(replace_causal_mask)
                transforms.append(replace_rms_norm_with_native_rms_norm)
                # pyre-fixme[16]: Module `backends` has no attribute `qualcomm`.
                transforms.append(convert_linear_to_conv2d)
            else:
                transforms.append(replace_kv_cache_with_simple_kv_cache)
                transforms.append(replace_sdpa_with_flex_sdpa)
                transforms.append(replace_causal_mask)
                transforms.append(replace_rms_norm_with_native_rms_norm)
                if args.optimized_rotation_path:
                    transforms.append(fuse_layer_norms
                    transforms.append(
                        get_model_with_r1_r2(args.optimized_rotation_path)
                    )
                # pyre-fixme[16]: Module `backends` has no attribute `qualcomm`.
                transforms.append(convert_linear_to_conv2d)
```



```python
def _export_llama(modelname, args) -> LLMEdgeManager:  # noqa: C901
    _validate_args(args)
    pt2e_quant_params, quantizers, quant_dtype = get_quantizer_and_quant_params(args)

    # export_to_edge
    builder_exported_to_edge = (
        prepare_for_llama_export(modelname, args)
        .capture_pre_autograd_graph()
        .pt2e_quantize(quantizers)
        .export_to_edge()
    )
```



```python
def _export_llama(modelname, args) -> LLMEdgeManager:  # noqa: C901
    _validate_args(args)
    pt2e_quant_params, quantizers, quant_dtype = get_quantizer_and_quant_params(args)

    # export_to_edge
    builder_exported_to_edge = (
        _prepare_for_llama_export(modelname, args)
        .capture_pre_autograd_graph()
        .pt2e_quantize(quantizers)
        .export_to_edge()
    )
```



```python
def _export_llama(modelname, args) -> LLMEdgeManager:  # noqa: C901
    _validate_args(args)
    pt2e_quant_params, quantizers, quant_dtype = get_quantizer_and_quant_params(args)

    # export_to_edge
    builder_exported_to_edge = (
        _prepare_for_llama_export(modelname, args)
        .capture_pre_autograd_graph()
        .pt2e_quantize(quantizers)
        .export_to_edge()
    )
```

pt2e_quantize : 양자화를 수행하는 함수

↓

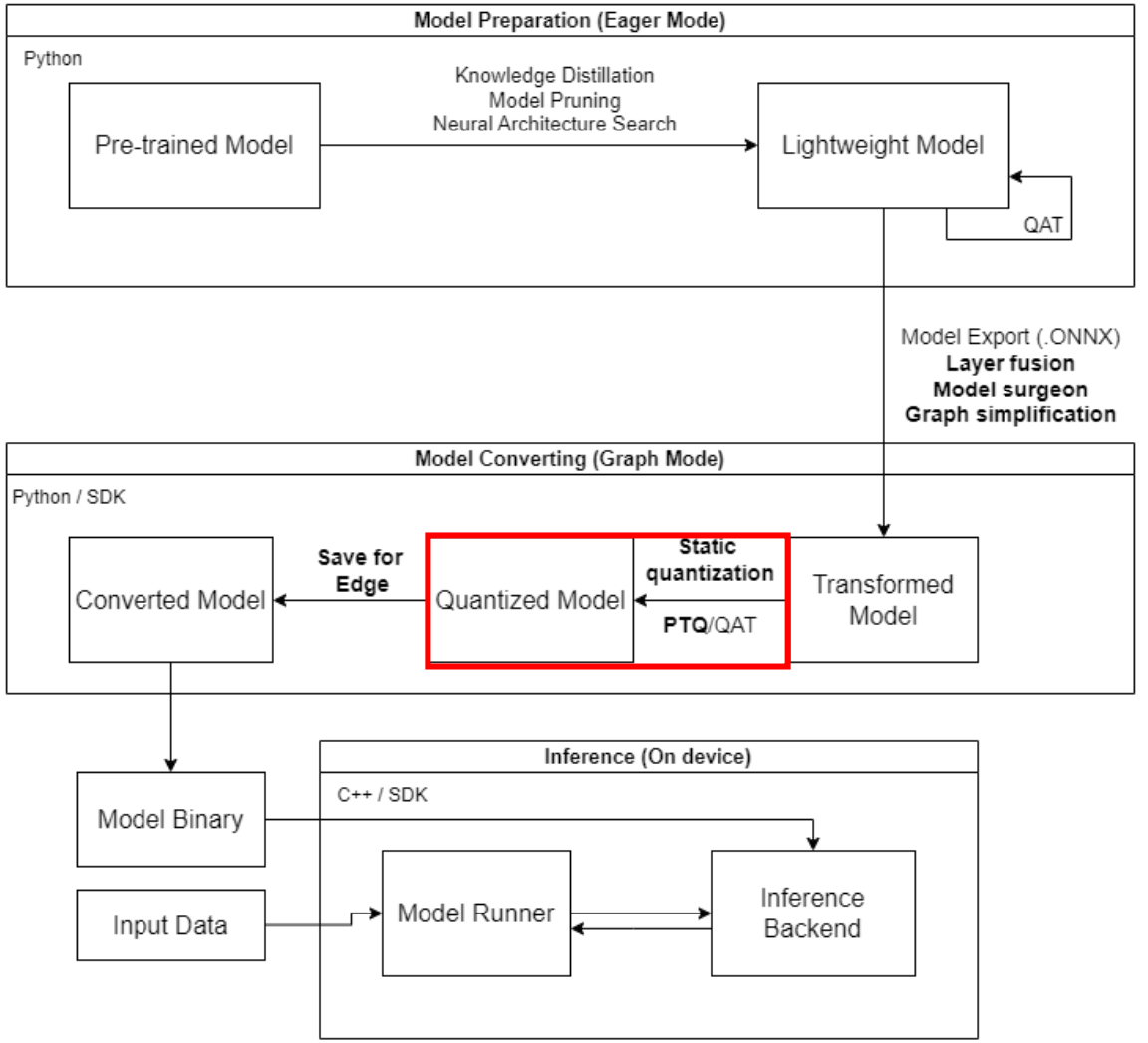
prepare_pt2e : Graph 모델 불러오기

↓

pt2e_calibrate : 양자화 범위 설정

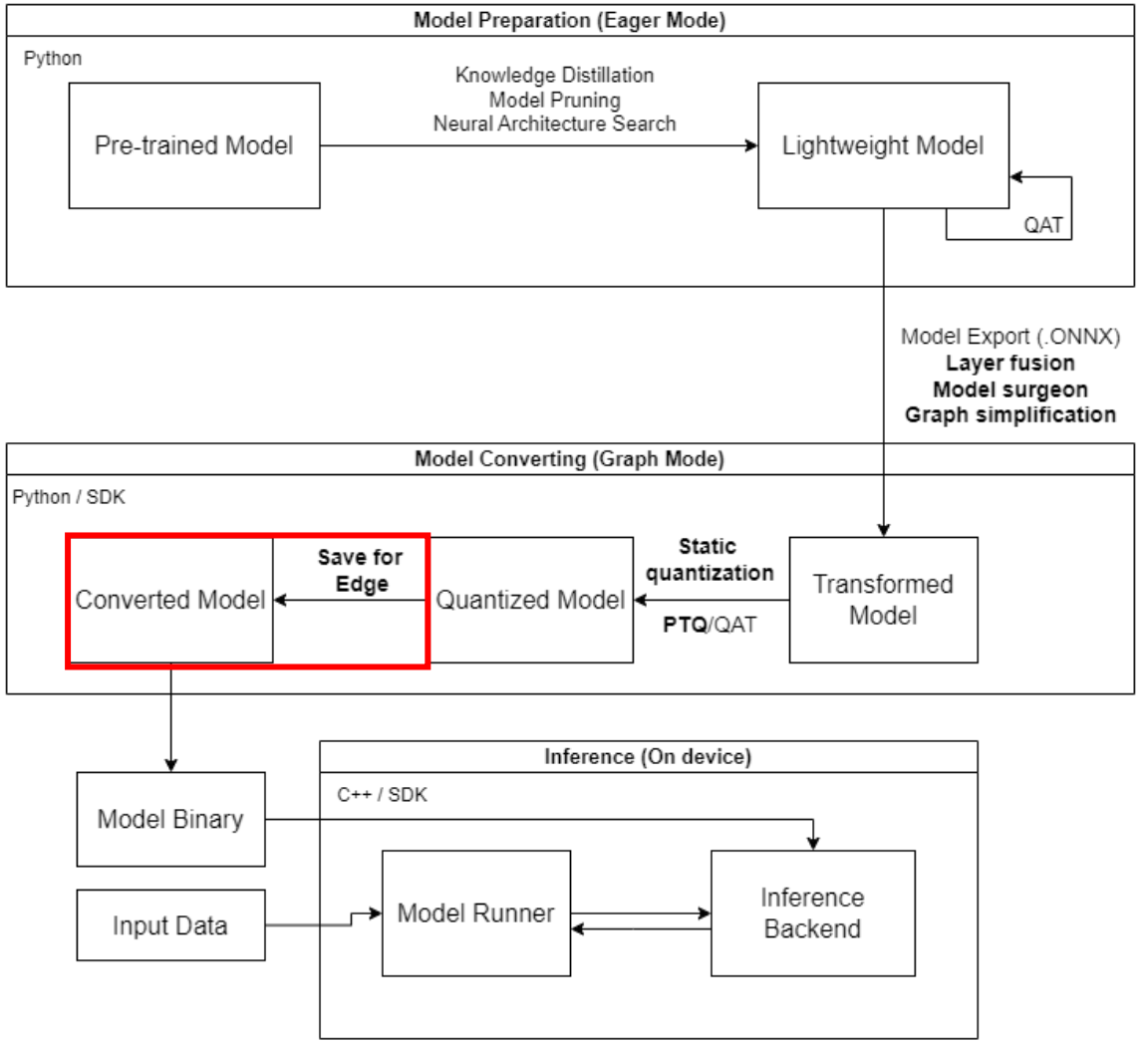
↓

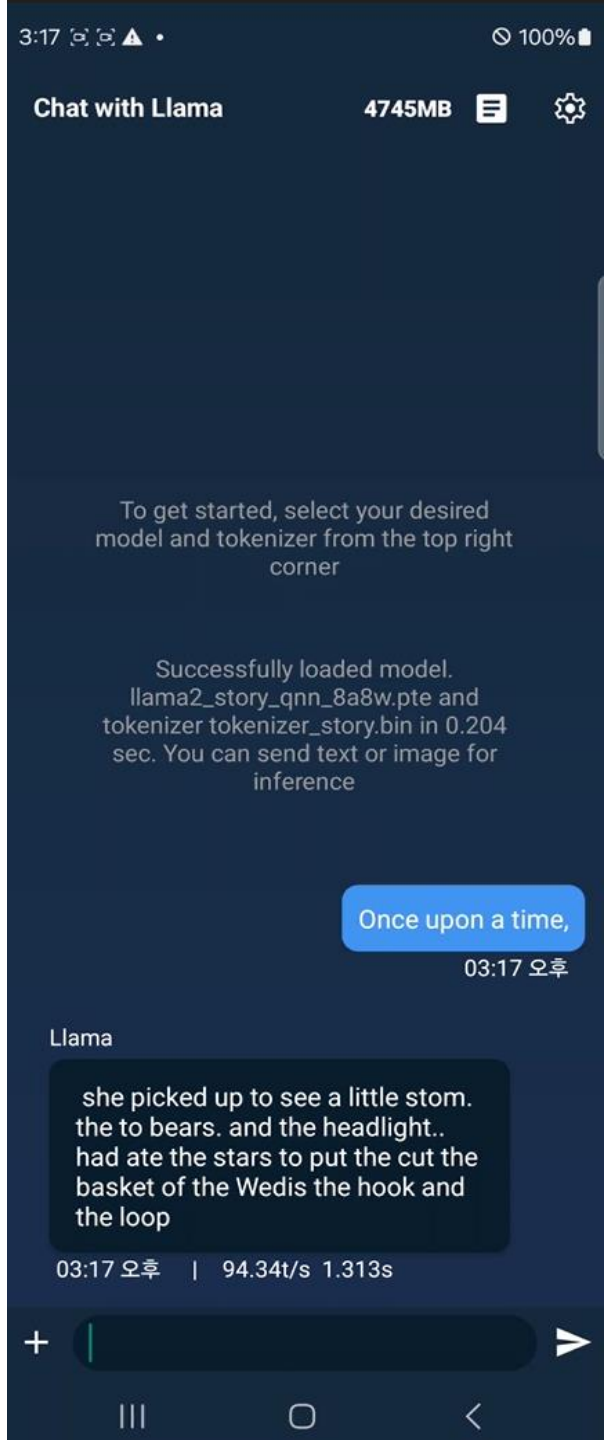
convert_pt2e : 양자화 수행

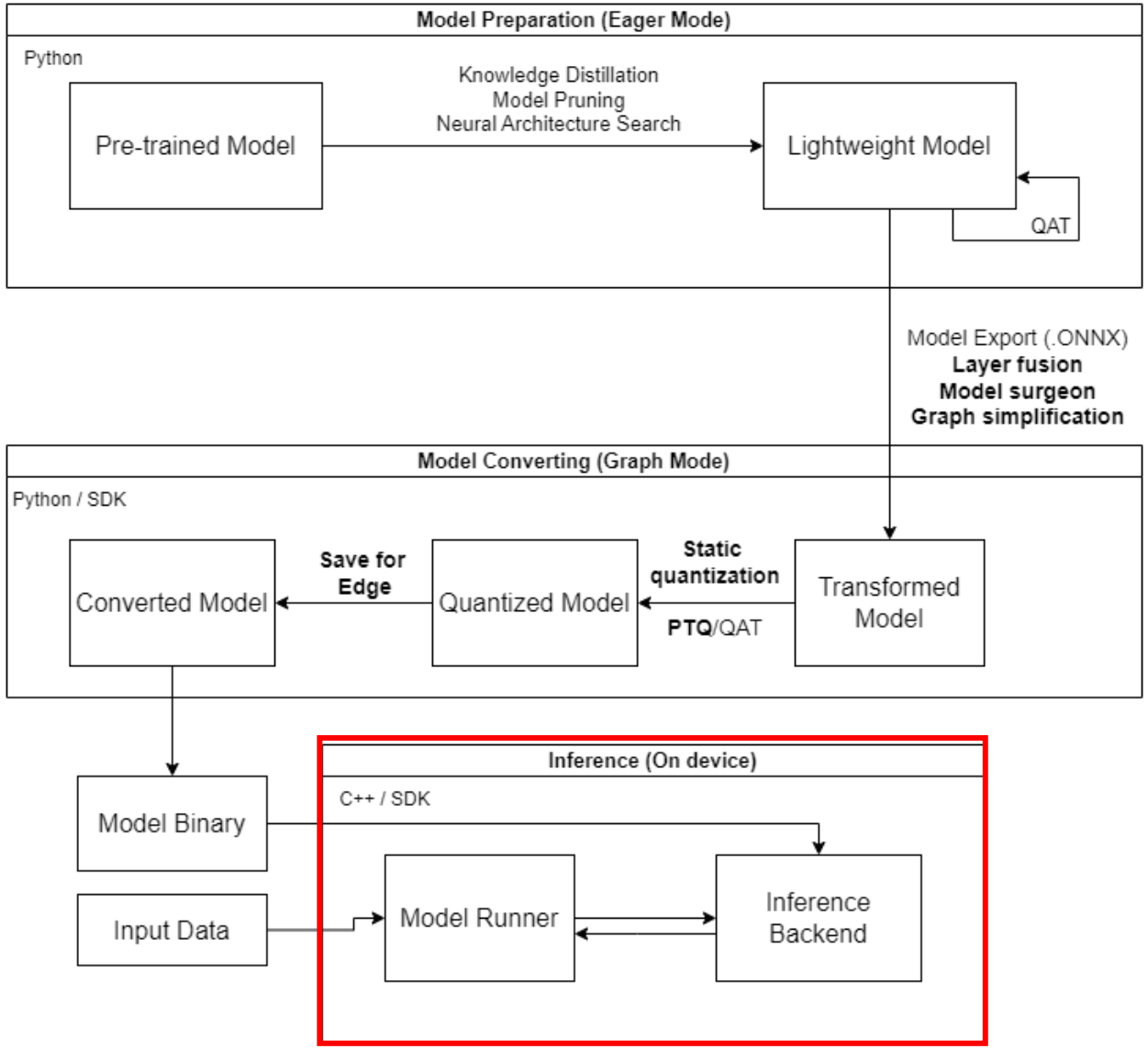


```python
def _export_llama(modelname, args) -> LLMEdgeManager:  # noqa: C901
    _validate_args(args)
    pt2e_quant_params, quantizers, quant_dtype = get_quantizer_and_quant_params(args)

    # export_to_edge
    builder_exported_to_edge = (
        _prepare_for_llama_export(modelname, args)
        .capture_pre_autograd_graph()
        .pt2e_quantize(quantizers)
        .export_to_edge()
    )
```

✓ 디바이스에 탑재할 수 있도록 모델 최종 변환




3:17
100%
Chat with Llama
4745MB
To get started, select your desired model and tokenizer from the top right corner
Successfully loaded model. llama2_story_qnn_8a8w.pte and tokenizer tokenizer_story.bin in 0.204 sec. You can send text or image for inference
Once upon a time,
03:17 오후
Llama
she picked up to see a little stom. the to bears. and the headlight.. had ate the stars to put the cut the basket of the Wedis the hook and the loop
03:17 오후 | 94.34t/s 1.313s


Model Preparation (Eager Mode)
Python
Pre-trained Model
Knowledge Distillation
Model Pruning
Neural Architecture Search
Lightweight Model
QAT
Model Export (.ONNX)
Layer fusion
Model surgeon
Graph simplification
Model Converting (Graph Mode)
Python / SDK
Converted Model
Save for Edge
Quantized Model
Static quantization
PTQ/QAT
Transformed Model
Model Binary
Input Data
Inference (On device)
C++ / SDK
Model Runner
Inference Backend

02
LLaMA-3.2-1B-Instruct

```
python -m examples.models.llama.export_llama \
    -t ${MODEL_DIR}/tokenizer.model \
    -p ${MODEL_DIR}/params.json \
    -c ${MODEL_DIR}/consolidated.00.pth  \
    --use_kv_cache  \
    --qnn \
    --pt2e_quantize qnn_16a16w \
    --disable_dynamic_shape \
    --calibration_tasks wikitext \
    --calibration_limit 1 \
    --calibration_seq_length 128 \
    --calibration_data "<|start_header_id|>system<|end_header_id|>\n\nYou are a funny
        chatbot.<|eot_id|><|start_header_id|>user<|end_header_id|>\n\nCould you tell me
        about Facebook?<|eot_id|><|start_header_id|>assistant<|end_header_id|>\n\n" \
    --metadata '{"get_bos_id":128000, "get_eos_ids":[128009, 128001]}' \
    --output_name="llama32_qnn_cal_16a16w.pte"
```

```
metadata = {
    "get_bos_id": 3 if is_fairseq2 else 1,
    "get_eos_ids": [3] if is_fairseq2 else [2],
    "get_max_seq_len": max_seq_len,
    "get_n_layers": n_layers,
    "get_vocab_size": vocab_size,
    "use_kv_cache": use_kv_cache,
    "use_sdpa_with_kv_cache": use_sdpa_with_kv_cache,
    "enable_dynamic_shape": enable_dynamic_shape,
}
```

```
{
  "id": 1,
  "content": "<s>",
  "single_word": false,
  "lstrip": false,
  "rstrip": false,
  "normalized": false,
  "special": true
},
{
  "id": 2,
  "content": "</s>",
  "single_word": false,
  "lstrip": false,
  "rstrip": false,
  "normalized": false,
  "special": true
}
```

기존 코드에는 LLaMA-2 기준으로 토큰이 설정되어 있음

```
"128000": {
  "content": "<|begin_of_text|>",
  "lstrip": false,
  "normalized": false,
  "rstrip": false,
  "single_word": false,
  "special": true
},
"128001": {
  "content": "<|end_of_text|>",
  "lstrip": false,
  "normalized": false,
  "rstrip": false,
  "single_word": false,
  "special": true
},
```

```
"128009": {
  "content": "<|eot_id|>",
  "lstrip": false,
  "normalized": false,
  "rstrip": false,
  "single_word": false,
  "special": true
},
```

LLaMA-3.2 모델에 맞게 토큰 수정

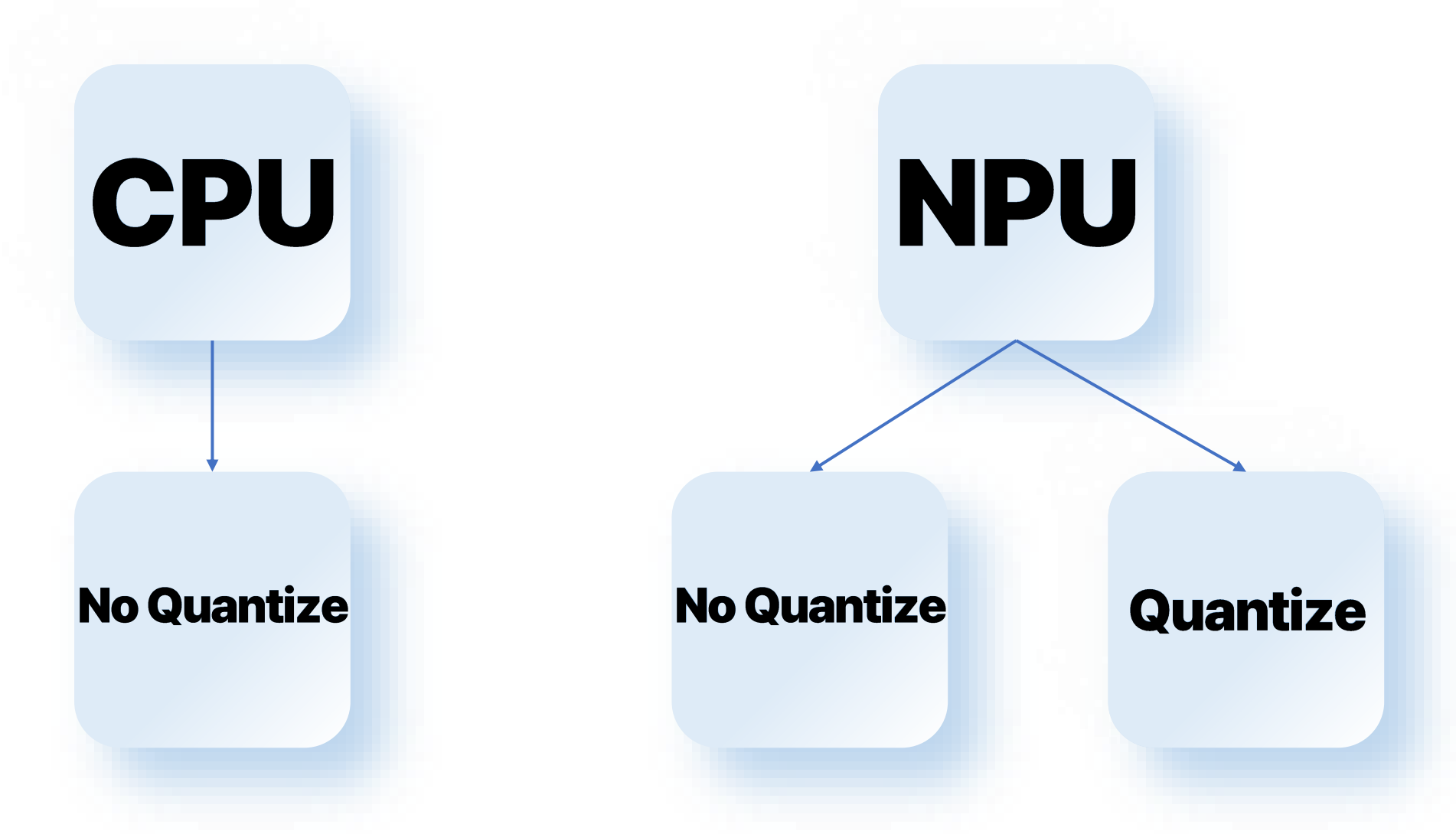
```
python -m examples.models.llama.export_llama \
    -t ${MODEL_DIR}/tokenizer.model \
    -p ${MODEL_DIR}/params.json \
    -c ${MODEL_DIR}/consolidated.00.pth  \
    --use_kv_cache  \
    --qnn \
    --pt2e_quantize qnn_16a16w \
    --disable_dynamic_shape \
    --calibration_tasks wikitext \
    --calibration_limit 1 \
    --calibration_seq_length 128 \
    --calibration_data "<|start_header_id|>system<|end_header_id|>\n\nYou are a funny
       chatbot.<|eot_id|><|start_header_id|>user<|end_header_id|>\n\nCould you tell me
       about Facebook?<|eot_id|><|start_header_id|>assistant<|end_header_id|>\n\n" \
    --metadata '{"get_bos_id":128000, "get_eos_ids":[128009, 128001]}' \
    --output_name="llama32_qnn_cal_16a16w.pte"
```

➢ Calibration 목적

- ✓ 양자화시 발생하는 성능 저하를 최소화 하도록 적절한 범위를 설정
- ✓ 입력 데이터 분포를 기반으로, Weight와 Activation의 Min, Max 값을 결정하여 양자화 수행
- ✓ Calibration이 잘못될 경우, 중요한 값들이 Clipping 되거나 잘못된 범위로 인해 양자화 오차 커져 성능 저하 발생

➢ Calibration Task & Data

- ✓ Task를 이용해 Weight와 Activation의 분포를 분석하여 양자화에 필요한 Scale과 Zero-point를 계산하고 범위 설정
- ✓ WikiText에 포함되어 있지 않은 스페셜 토큰을 포함시켜 실제 입력 데이터의 특징을 반영


CPU
No Quantize
NPU
No Quantize
Quantize

03
Tokenizer Code Review

Tokenizer

Tokenzier 종류

- Word-based Tokenizer
  - ✓ 단어 단위로 텍스트를 분리
  - ✓ Vocab size 매우 커짐
  - ✓ GPT-1 Tokenizer
- Subword-based Tokenizer
  - ✓ 단어를 더 작은 단위로 분리
  - ✓ GPT-2, BERT 등에서 사용
- Byte-based Tokenizer
  - ✓ 문자를 UTF-8 바이트로 변환
  - ✓ 고정된 어휘 크기
  - ✓ GPT-3 Byte-level BPE

최신 Tokenizer

- HuggingFace Tokenizer
  - ✓ Rust로 구현되어 Python보다 빠름
  - ✓ BPE, WordPiece, Unigram 등 지원
- SentencePiece
  - ✓ 언어 독립적, 텍스트를 사전 전처리 없이 처리 가능
  - ✓ 높은 압축 효율
- TikTokenizer
  - ✓ Byte-level BPE를 사용함
  - ✓ Rust로 구현되어 빠른 속도 보임

GPT2 Tokenizer

Tokenizer

Tokenizer?

Merge text : 단어의 우선 순위

Vocab json : 단어가 어느 값일지

: string to number

: number to string

```
void parse_merge_file(std::string const& merges_file);
void parse_vocab_file(std::string const& vocab_file);
```

Tokenizing part

Byte Encoder

Byte Decoder

Byte pair Encoding

BPE Ranks

```
// A Regular expression that handles text tokenization
static constexpr std::string_view pattern{
        "'s|'t|'re|'ve|'m|'ll|'d| ?\\p{L}+| ?\\p{N}+| ?[^\\s\\p{L}\\p{N}]+|\\s+(?!\\S)|\\s+"};
```

Mapping part

Encoder

Decoder

- 축약형 분리 (“ ‘s, ‘ll, ‘t” 등)
- 공백이 앞에 올 수 있는 단어 및 숫자 처리 (“[PAD]Hello” == “Hello”)
- 문자와 숫자를 개별 토큰으로 분리
- 특수 문자, 구두 점을 개별 토큰으로 분리
- N개의 공백도 별도의 토큰으로 분리

GPT2 Tokenizer

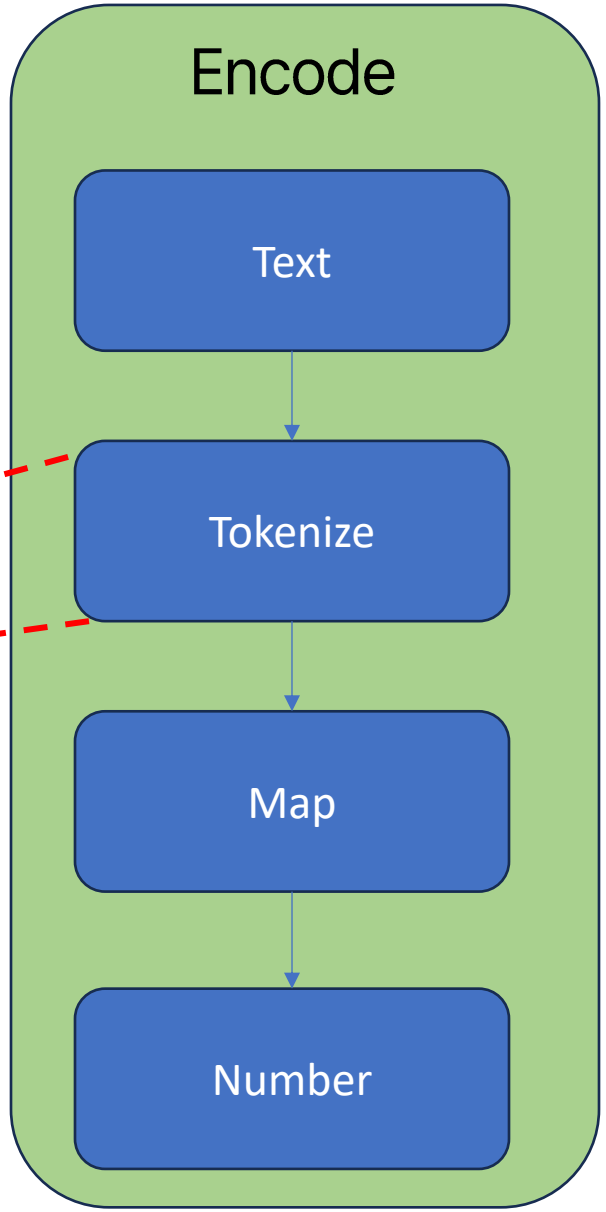
Tokenizer

Byte Encoder | Byte Decoder | Encoder | Decoder

Hello Hello World!

```
std::unordered_map<std::string, std::uint64_t> m_encoder;
std::unordered_map<std::uint64_t, std::string> m_decoder;

std::unordered_map<char, std::string> m_bytes_encoder;
std::unordered_map<std::string, char> m_bytes_decoder;
```

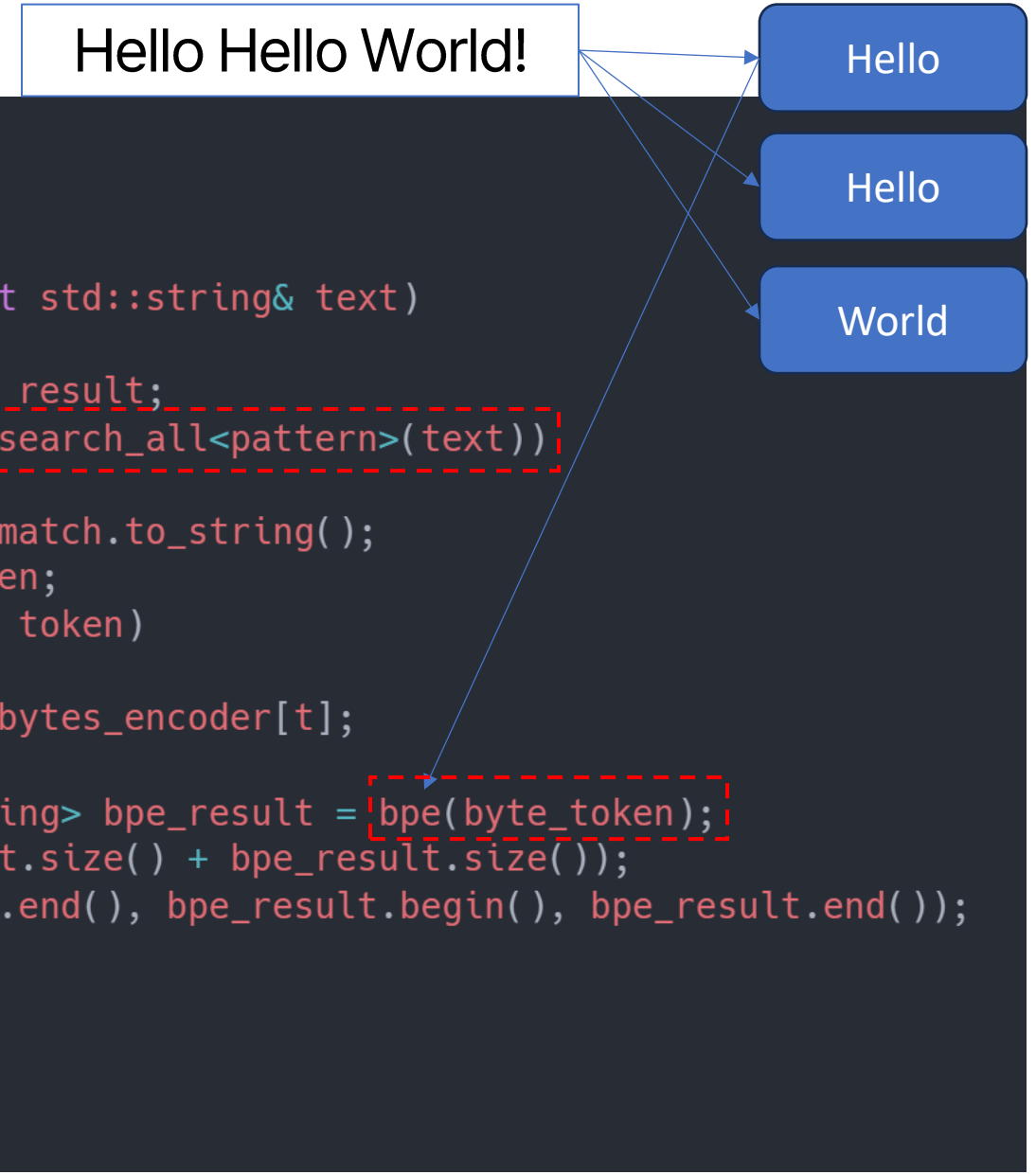
```
std::vector<int64_t>
GPT2Tokenizer::encode(const std::string& text)
{
    std::vector<std::string> tokens = tokenize(text);
    std::vector<int64_t>     token_ids;
    token_ids.reserve(tokens.size());
    std::transform(tokens.begin(),
                   tokens.end(),
                   std::back_inserter(token_ids),
                   [this](const std::string& token) { return m_encoder[token]; });
    return token_ids;
}
```

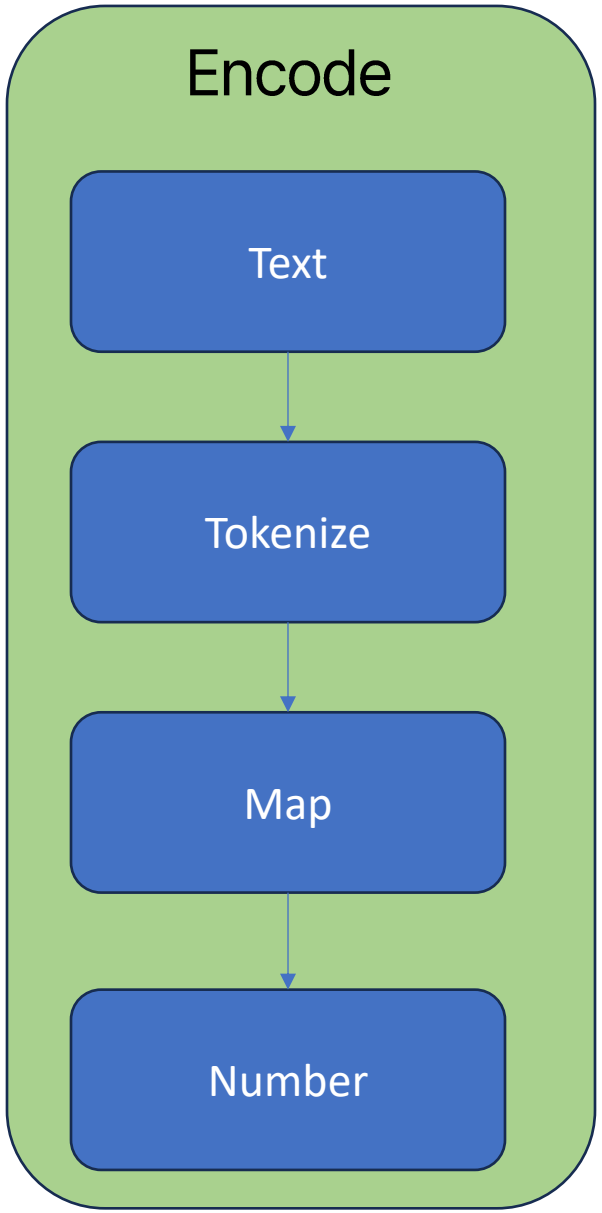


GPT2 Tokenizer

Tokenizer

```cpp
std::vector<std::string>
GPT2Tokenizer::tokenize(const std::string& text)
{
    std::vector<std::string> result;
    for (auto match : ctre::search_all<pattern>(text))
    {
        std::string token = match.to_string();
        std::string byte_token;
        for (const auto& t : token)
        {
            byte_token += m_bytes_encoder[t];
        }
        std::vector<std::string> bpe_result = bpe(byte_token);
        result.reserve(result.size() + bpe_result.size());
        result.insert(result.end(), bpe_result.begin(), bpe_result.end());
    }

    return result;
}
```





GPT2 Tokenizer

Byte Pair Encoding (Preprocess)

Hello

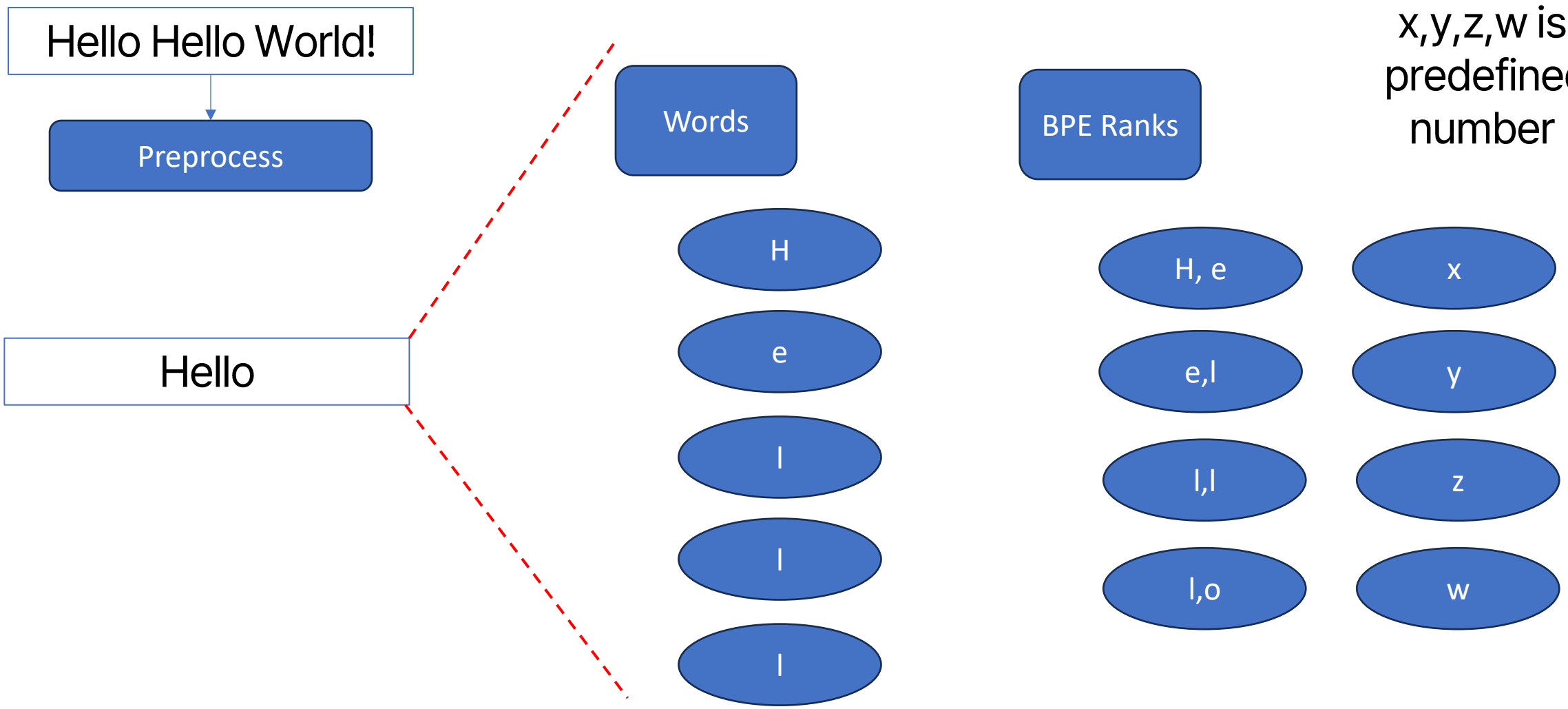
```cpp
std::vector<BPERanks::const_iterator> ranks;
// 벡터 word: 입력 문자열을 각 코드포인트(문자)로 나눈 결과를 저장
std::vector<std::string> word;
ranks.reserve(token.size() - 1); // ranks의 용량 예약: 최대 bigram 개수 (문자 수 - 1)
word.reserve(token.size());      // word의 용량 예약: 최대 문자 수

// 입력 문자열을 코드포인트 단위로 나누고 bigram 순위를 계산하는 부분
{
    size_t i = 0;
    while (true)
    {
        // 현재 코드포인트의 길이를 구함
        int length = codepoint_length(token[i]);
        // 다음 코드포인트의 길이를 구함
        int next_length = codepoint_length(token[i + length]);
        // 현재 문자와 다음 문자의 bigram 순위를 ranks에 추가
        ranks.push_back(
            m_bpe_ranks.find({token.substr(i, length), token.substr(i + length, next_length)}));
        // 현재 문자를 word에 추가
        word.push_back(token.substr(i, length));
        i += length;                    // 인덱스를 다음 문자로 이동
        if (i >= token.size()) break; // 문자열 끝에 도달하면 루프 종료
        if (i + next_length >= token.size())
        {
            // 마지막 문자를 word에 추가하고 종료
            word.emplace_back(token.substr(i, next_length));
            break;
        }
    }
}
```

BPE Ranks

```cpp
size_t
codepoint_length(const char c)
{
    if ((c & 0xf8) == 0xf0)
        return 4;
    else if ((c & 0xf0) == 0xe0)
        return 3;
    else if ((c & 0xe0) == 0xc0)
        return 2;
    else
        return 1;
}
```

GPT2 Tokenizer

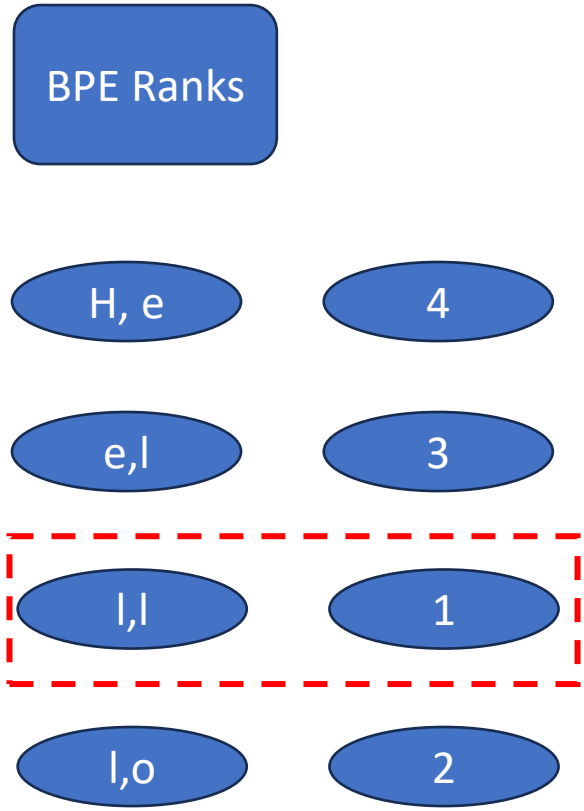
Byte Pair Encoding (Preprocess)



GPT2 Tokenizer

Byte Pair Encoding (Encoding)

```cpp
// ranks에서 가장 낮은 순위를 가진 bigram을 찾음
const auto bigram = std::min_element(
    ranks.begin(), ranks.end(), [this](const auto& lhs, const auto& rhs) -> bool {
        if (lhs == m_bpe_ranks.end() && rhs == m_bpe_ranks.end())
        {
            return false; // 둘 다 끝이면 false 반환
        }
        else if (lhs == m_bpe_ranks.end() || rhs == m_bpe_ranks.end())
        {
            return (lhs != m_bpe_ranks.end()); // 하나가 끝이면 유효한 다른 하나 선택
        }
        else
        {
            return lhs->second < rhs->second; // 둘 다 유효하면 순위가 낮은 것을 선택
        }
    });

if (bigram == ranks.end() || *bigram == m_bpe_ranks.end())
{
    // 더 이상 병합할 bigram이 없다면 종료
    break;
}
```



GPT2 Tokenizer

Byte Pair Encoding (Encoding)

H,e,l,l,o

Hello

For 1st iteration

l,l

H, e

l,l -> ll

```cpp
const auto [first, second] = (*bigram)->first; // 병합할 bigram의 첫 번째와 두 번째 문자
std::vector<std::string> new_word;             // 새로운 단어 벡터 생성

size_t i = 0;
while (i < word.size())
{
    // 첫 번째 문자를 찾음
    const auto wordIterator = std::find(word.begin() + i, word.end(), first);
    if (wordIterator == word.end())
    {
        // 첫 번째 문자가 없으면 나머지 문자를 모두 new_word에 추가하고 종료
        std::copy(word.begin() + i, word.end(), std::back_inserter(new_word));
        break;
    }

    // 첫 번째 문자 이전의 모든 문자를 new_word에 추가
    std::copy(word.begin() + i, wordIterator, std::back_inserter(new_word));
    i = std::distance(word.begin(), wordIterator); // 현재 인덱스를 업데이트

    if (word[i] == first and i < word.size() - 1 and word[i + 1] == second)
    {
        // 첫 번째 문자 다음에 두 번째 문자가 있으면 병합하여 new_word에 추가
        new_word.push_back(first + second);
        i += 2; // 두 문자를 병합했으므로 인덱스를 두 칸 이동
    }
    else
    {
        // 병합하지 않는 경우 현재 문자를 new_word에 추가
        new_word.push_back(word[i]);
        i += 1; // 인덱스를 하나 증가
    }
}
word = std::move(new_word); // word를 new_word로 업데이트
```

GPT2 Tokenizer

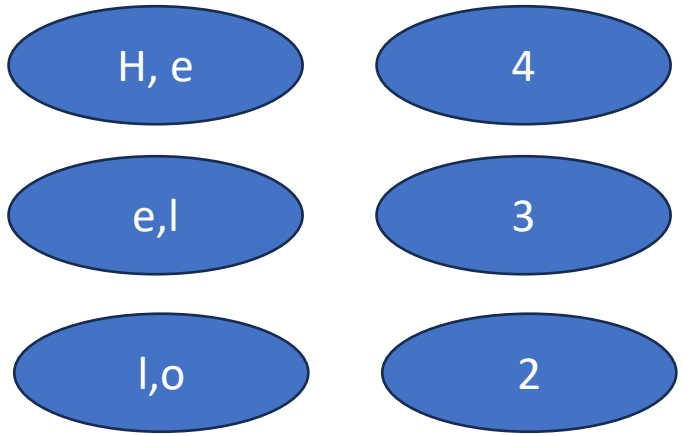
Tokenizer

Remove merged bigram

```
if (word.size() == 1)
    break; // word의 크기가 1이면 더 이상 병합할 수 없으므로 종료
else
{
    // 새로운 bigram 순위를 ranks에 업데이트
    for (size_t i = 0; i < word.size() - 1; ++i)
    {
        ranks[i] = m_bpe_ranks.find({word[i], word[i + 1]});
    }
    ranks.resize(word.size() - 1); // ranks의 크기를 업데이트
}
```

BPE Ranks

x,y,z,w is predefined number



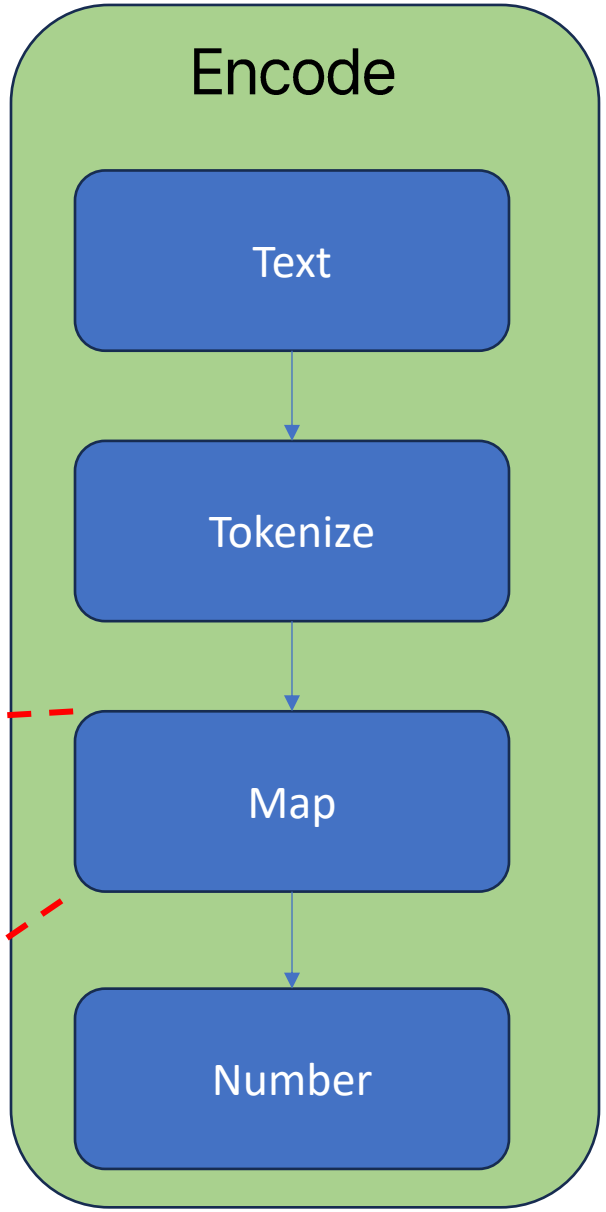
GPT2 Tokenizer

Tokenizer

Byte Encoder | Byte Decoder | Encoder | Decoder

```
std::unordered_map<std::string, std::uint64_t> m_encoder;
std::unordered_map<std::uint64_t, std::string> m_decoder;

std::unordered_map<char, std::string> m_bytes_encoder;
std::unordered_map<std::string, char> m_bytes_decoder;
```

```
std::vector<int64_t>
GPT2Tokenizer::encode(const std::string& text)
{
    std::vector<std::string> tokens = tokenize(text);
    std::vector<int64_t>     token_ids;
    token_ids.reserve(tokens.size());
    std::transform(tokens.begin(),
                   tokens.end(),
                   std::back_inserter(token_ids),
                   [this](const std::string& token) { return m_encoder[token]; });
    return token_ids;
}
```



End.